Link2Link: A Robust Probabilistic Routing Algorithm via Edge-centric Graph Reinforcement Learning

Jingli Zhou, Yuqian Song*, Xinyuan Li, Wenli Zhou, Jun Liu
School of Artificial Intelligence, Beijing University of Post and Telecommunications, Beijing, China
Emails: {wayzer, songyuqian, xinyuanli, zwl, liujun}@bupt.edu.cn

***Abstract*—As network services become more complex, efficient routing has become crucial for ensuring end-user satisfaction. To address this challenge, researchers are increasingly turning to routing algorithms that integrate Graph Neural Networks (GNNs) with Deep Reinforcement Learning (DRL), leveraging the natural graph structure of network topologies. However, a significant challenge with existing algorithms is their inability to generalize across different topologies without requiring retraining, a constraint that is impractical in real-world applications. To overcome this limitation, we propose a novel GNN-DRL-based routing algorithm, Link2Link, designed to decouple DRL-learned knowledge from specific network topologies by focusing on link-level features. Extensive experiments demonstrate that Link2Link achieves robust performance across diverse topologies, consistently outperforming OSPF without requiring retraining, making it a scalable and adaptable solution for modern network routing challenges.**

***Index Terms*—Network Routing Algorithm, Deep Reinforcement Learning, Graph Neural Network, Robustness**

I. Introduction

The increasing network service demands, driven by the rise of applications like 5G, IoT, and cloud computing, has introduced significant challenges for traditional routing algorithms. Algorithms such as shortest-path routing, which rely on static topologies and pre-configured rules, struggle to cope with the dynamic and unpredictable nature of modern networks. These limitations become even more pronounced in large-scale, heterogeneous environments. As a result, there is a growing need for more advanced routing algorithms that can adapt to fluctuating traffic patterns, ensure efficient resource utilization, and scale effectively across diverse network conditions.

In recent years, deep reinforcement Learning (DRL) [1] has gained considerable attention in network routing optimization [2]–[4]. Due to its experience-driven real-time decision-making capabilities, it excels in dynamic real-world networks. Furthermore, researchers have increasingly incorporated Graph Neural Networks (GNNs) [5] into DRL-based routing algorithms to better capture the complex relationships within network topologies inherent in network topologies, as these are naturally abstracted as graphs [6]–[8].

Despite these advancements, existing algorithms still face unresolved issues and challenges. For instance, the work in [6] concatenated GAT-aggregated node features and feeds them into the policy network, resulting in a fixed actor network dimension that cannot be generalized across different network scales. To relieve the dependence on the global graph information, GAPPO [8] proposes concatenating two node features as link features, and the actor calculates the weight of each link separately. In this way, the dimensions of the actor can remain unchanged with varying network scales. However, this approach struggles with the extraction of relevant information from node features, as multiple link details are embedded within the same node feature in an uncertain order. Consequently, the actor implicitly learns the relationships between nodes and links, making it dependent on the specific topology.

To overcome the limitations of existing approaches, we propose a novel DRL-based routing algorithm called Link2Link, which shifts the focus from network nodes to links, recognizing that the core of routing lies in selecting links for packet transmission based on their states. Unlike node-based features, which depend on the number or order of adjacent links, the feature dimension for link states remains constant across different topologies as each link is associated only with itself and its two endpoints. This homogeneity endows the Link2Link algorithm with significant robustness. Furthermore, the network topology is fed into Graph Neural Networks (GNNs) in a specialized manner, allowing the agent to learn topology-independent link features that guide link weight decisions. This method fosters the development of a more robust and generalizable routing strategy that can be trained once and efficiently applied across diverse network topologies.

Building on this approach, our contributions in this work include:

- We extend the probabilistic routing protocol to accommodate more realistic scenarios involving multi-source and multi-destination traffic.
- We introduce the Link2Link algorithm, which shifts the focus entirely to links. Our approach leverages a fully edge-centric GNN aggregation mechanism, enabling the computation of link weights directly from link features rather than relying on node-based characteristics.
- We conducted extensive experiments to validate the performance of Link2Link, across various network topologies. We investigated the impact of different GNN layer depths on the algorithm's effectiveness. Notably, experi-

This work is partially supported by the National Natural Science Foundation of China under the Grant No. 62371057. *(*Corresponding author: Yuqian Song)*

mental results show that Link2Link performs well even on previously unseen, larger topologies without the need for retraining, underscoring its robustness and generalizability.

II. Methods

A. System Model

The network is conceptualized as a directed graph $G = (N, E)$, where N and E represent finite sets of nodes and links respectively. Packets are allowed to enter the network from any nodes, be forwarded to adjacent nodes, and proceed to any designated destination.

Each link $e = (N_i, N_j)$, represents a directed edge in graph G from node N_i to node N_j, with link capacity $B(e)$, propagation delay $K(e)$. And in the sender node, there is a queue buffer for each link with the capacity "QueueSize", using a first-in-first-out (FIFO) discipline.

A packet p starts its journey from the source node $S(p)$ at time $t_0(p)$ with size $L(p)$, targeting the destination node $D(p)$. On the way, it needs to queue up and pass through several links, which form the path $P(p)$. The queuing delay is denoted as $T\text{"}queue\text{"}(e,p)$. So the delay for one packet is defined as

$$T(p) = \sum_{e \in P(p)} \left[T_{\text{queue}}(e,p) + K(e) + \frac{L(p)}{B(e)} \right]. \tag{1}$$

The processing delay of the switch is disregarded, as it's negligible small as compared to the queuing delay.

In addition, a packet is considered successfully delivered when reaches the destination within the survival time "SurviveTime" and the maximum hop count "TTL", marked as $Success(p) = 1$. Otherwise, it's considered lost, where $Success(p) = 0$.

For convenience, the statistical interval is set as "StepTime". As a result, the average delay of a statistical interval is

$$\text{delay}_{\text{avg}} = \frac{\sum_{p \in \text{Delivered}} T(p)}{|\text{Delivered}|}, \tag{2}$$

where "Delivered" denotes the set of packets sent within the interval and finally reach their destination node, and $abs(dot.c)$ denotes the cardinality of the packet set. Besides, the loss rate is defined as

$$\text{loss} = 1 - \frac{|\text{Delivered}|}{|\text{All}|}, \tag{3}$$

where "All" denotes the set of all packets sent in the statistical interval.

The optimization objective of the network routing problem is to determine the paths of packets to enhance better QoS metrics, typically measured by delay and packet loss rate. While the TTL-based packet loss design from the IP protocol is retained, the main cause of packet loss is link congestion, which causes queue overflow or data packet transmission timeouts. According to (1), the delay is mainly affected by the number of hops and queuing time. Therefore, the agent should avoid congestion in the network and choose the possible paths with fewer hops for the data packets.

B. Key Element Design

1) State: The state is a set of link features that can reflect the information of the destination, flow, and link. We employ link utilization as the main feature, as it reflects traffic demand, the congestion and residual capacity of the link, as well as the quality of the routing policies. The link utilization for destination d and link e is calculated by

$$u(d,e) = \frac{\sum_{p \in P(d,e)} L_p}{B(e) \times \text{StepTime}}, \tag{4}$$

where $P(d,e)$ represents the packets pass through link e and target destination d inside the statistical interval. The total link utilization for all destinations is

$$u_G(e) = \sum_{d \in D} u(d,e). \tag{5}$$

In addition, we employ the distance difference between the nodes at both ends of the link to indicate the destination, which is defined as

$$\Delta D(d,e) = D(v,d) - D(u,d) \in \{-1, 0, 1\}, \tag{6}$$

where u and v represent the nodes at both ends of the link e and $D(v,d)$ is the distance from node N_v to node N_d in the directed graph.

To sum up, the state for destination N_d and link e consists of three parts: link utilization, total link utilization, and destination indication. It's defined as

$$s_{d,e} = [u_G(e), u(d,e)] \| \text{onehot}[\Delta D(d,e)], \tag{7}$$

where $X \| Y$ represents the concatenation of X and Y, and "onehot" means one-hot encoding. The complete state contains the link features for each link for each destination, as

$$S = \{s_{d,e}\}_{D \times E}. \tag{8}$$

2) Actions: The action space comprises weights for the destination-differentiated probabilistic routing protocol, defined as

$$A = W_{D \times E} = \{w_{d,e} \mid d \in D, e \in E\}, \tag{9}$$

where $w_{d,e}$ is the weight for each directed link e per destination d. What follows is the detailed design of the action.

The probabilistic routing protocol, where nodes make forwarding decisions based on probabilities derived from weights, provides a flexible and adaptive approach to the Network Routing Problem. The probabilistic decisions at each node enhance exploratory capabilities, enabling the development of more sophisticated routing strategies. Moreover, its simplicity facilitates implementation within switches in both simulation environments and SDN.

In previous work [4], each directed links are assigned one weight value, and switching nodes normalized these weights into probabilities to select an output link for packet forwarding. But it's not enough in general scenarios with multiple destinations. As illustrated in Fig. 1, considering a simple single-line topology, if there are flows only head towards a single

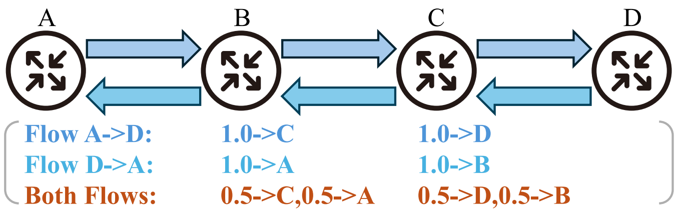

Fig. 1. Link Weight in a Single-Line Topology with Varied Flow Demands.

destination, switches could assign a link weight of 1.0 in that direction. However, in typical network communications with bidirectional flows, switches B and C encounter challenges in assigning appropriate link weights. For instance, setting equal weights of 0.5 for both directions can lead to looping traffic between B and C, resulting in increased delays and potential link congestion.

To address this, we propose an extension that considers the destination of the packets, where each destination is associated with a set of weight values. For the network $G = (N, E)$, the total amount of destination-specific probabilistic routing parameters is $D \times E$, with D ($D <= N$) representing the number of destinations. The extended process is mathematically expressed as

$$P(e \mid p) = \frac{w(D(p), e)}{\sum_{e' \in O(N_i)} w(D(p), e')}, \forall e \in O(N_i), \tag{10}$$

where $P(e \mid p)$ denotes the probability of forwarding the packet p, and $w(D(p), e)$ is the weight of link e for the destination $D(p)$.

3) Reward: According to the optimization goal, we prioritize loss rate over latency, but they sometimes conflict, e.g., more hops can reduce congestion but increase delay. Considering realistic needs, we assume that packets will retransmit if not successfully received after time "$timeout$", the utility delay satisfies

$$\text{delay}_u = \text{delay}_{\text{avg}} \cdot (1 - \text{loss}) + (\text{timeout} + \text{delay}_u) \cdot \text{loss}. \tag{11}$$

Extract "$delay$"$_u$ to the left side, the expression of utility delay is obtained as:

$$\text{delay}_u = \text{delay}_{\text{avg}} + \text{timeout} \cdot \text{loss}/(1 - \text{loss}). \tag{12}$$

The reward is set to negative utility delay, defined as

$$r = -(\text{delay}_{\text{avg}} + \text{timeout} \cdot \text{loss}/(1 - \text{loss})). \tag{13}$$

C. Link2Link Algorithm

The kernel of our proposed algorithm, as previously outlined, is to use link state to independently infer link weights, mapping the relationship between link states and link weights:

$$s_{d,e} \longrightarrow w_{d,e}. \tag{14}$$

In this case, instead of nodes, edges are used as the aggregation object of GNNs, and the connection relationship between links is input into GNN as edge index. Therefore, the topological connection relationship is only in the input, and

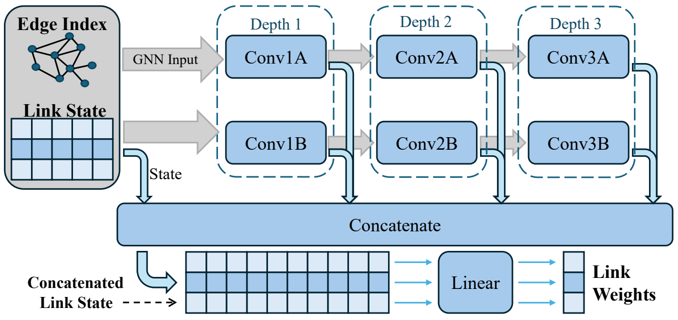

Fig. 2. Overview of the Link2Link Algorithm.

it does not need to be spliced once at the actor like GAPPO. Therefore, GNN is completely topologically decoupled and can be widely used in multiple topologies.

The model structure is illustrated in Fig. 2. The inputs include the edge index and the link state matrix, where each row corresponds to the state of a link. The edge index, representing the connectivity of the links, is utilized by the GNN convolution layer. The input undergoes multiple GNN convolution layers, for aggregated link states. Finally, all link states are concatenated and fed into a linear layer, which independently processes each link to derive link weight from features. Despite the apparent complexity of the linear layer's input, it is essentially a singular layer evaluating a single link's state and the output is one link weight value, emphasizing that the essence of the Link2Link algorithm resides in the links aggregation.

The goal of the links aggregation is to aggregate the features of adjacent links. To implement this, all links are treated as graph nodes (vertices) within the GNN, equating the graph's vertex count to the network's link count. The vertices features are represented as

$$\text{Vertices Features} = \begin{pmatrix} \text{Link State}_1 \\ \text{Link State}_2 \\ \vdots \\ \text{Link State}_E \end{pmatrix}_E . \tag{15}$$

To matching features, the links' head-to-tail connectivity is considered the graph's edges. As depicted in Fig. 3, links a, b, d are incoming to the switching node A, with c as the outgoing link, forming three edges $a, b, d \rightarrow c$ in the graph.

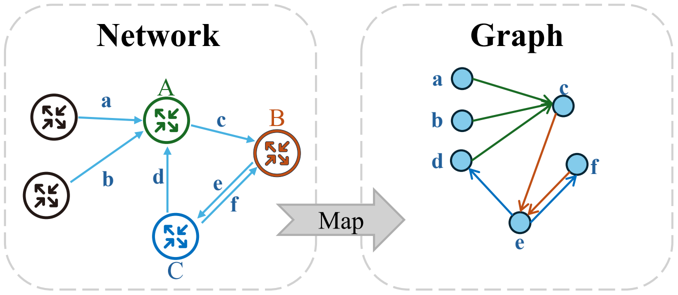

Fig. 3. Mapping Network Topology to a Graph Representation of Links.

Similarly, for node B, edges $c, f \rightarrow e$ exist, and for node C, edges $c \rightarrow d, f$ are present. This example maps a network topology of five nodes and six unidirectional links to a graph of six vertices and seven edges.

In addition, GNN aggregation is directed, so two sets of GNNs with different directions are employed, to learn the link information in each direction, as Conv1A, Conv1B, etc in Fig. 2. For instance, aggregation along the link direction can ascertain the total traffic volume moving toward the target, while aggregation in the reverse can gather insights into the link's remaining capacity.

D. Learning Algorithm

To train the model, the state-of-the-art Proximal Policy Optimization (PPO) [9] is utilized. To adapt PPO for our policy network, we introduce two additional components: a probabilistic policy and a critic network.

We adopt the Beta distribution as our probabilistic strategy to output link weights. Specifically, the linear layer's output in Fig. 2 is transformed into two parameters, α and β, of the Beta distribution with the mapping softplus$(\cdot) + 1$ to make it unimodal. Then, the weight values are sampled from the distribution, as depicted in Fig. 4. The Beta distribution is a continuous probability distribution defined on the interval $[0, 1]$, parameterized by two positive shape parameters, α (alpha) and β (beta), which control the shape of the distribution. The probability density function (PDF) of the Beta distribution is

$$f(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, \tag{16}$$

where $B(\alpha, \beta)$ is the Beta function, ensuring the total probability integrates to 1. The distribution is unimodal when $\alpha > 1$ and $\beta > 1$.

For the critic network, a Multi-Layer Perceptron (MLP) is employed, taking the concatenated link state matrix as input. As shown in Fig. 4, we initially process each link features separately with a linear layer to reduce the dimensionality to a smaller number (we chose 8). Then the reduced link state matrix is input into another MLP to estimate the state value. Because the critic network is only used in the training phase, it does not affect the robustness of the algorithm.

Model parameters are updated at fixed intervals using batch gradient updates, where each optimization epoch utilizes the advantages $A(s, a)$ to approximate rewards. In this work, we forego the discount factor of action value ($\gamma = 0$), as the network routing problem involves infinite steps and the reward already encapsulates the QoS metrics. The advantages are computed using value function estimates:

$$A(s_t) = r_t - V_\Phi(s_t), \tag{17}$$

where r_t is the reward and $V_\Phi(s_t)$ is derived from the value network.

The target policy parameters θ are refined by minimizing the PPO objective function, comprising a policy loss $cal(L)_p$, a value function loss $cal(L)_v$, and an entropy regularization loss $cal(L)_e$.

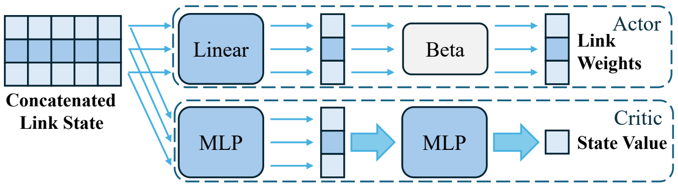

Fig. 4. Data Flow Diagram for Actor and Critic.

Let $r_t(theta)$ denotes the probability ratio as

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \tag{18}$$

where $theta$ represents the current policy parameters and $theta_"old"$ the parameters prior to the update. The surrogate objective is

$$\mathcal{L}_p = \mathbb{E}_t \left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right], \tag{19}$$

where $epsilon$ is a hyper-parameter, the second term clip$(r_t(\theta), 1-\epsilon, 1+\epsilon)$, involving the clipping of the probability ratio, modifies the surrogate objective by restricting the range of the probability ratio.

The value function loss is a squared-error loss defined as

$$\mathcal{L}_v = (V_\Phi(s_t) - r_t)^2, \tag{20}$$

The entropy regularization loss is defined as

$$\mathcal{L}_e = S_{\pi_\theta}(s_t), \tag{21}$$

where S denotes an entropy bonus.

Combining these terms, the combined objective is

$$\mathcal{L}_{p+v+e} = c_1\mathcal{L}_p + c_2\mathcal{L}_v + c_3\mathcal{L}_e, \tag{22}$$

with c_1, c_2, c_3 as coefficients.

In addition, to increase the speed of training, multiple worker threads interact with the environment in parallel, pooling their experiences into a shared buffer. Once the buffer reaches a predefined threshold, all workers are paused, and the agent parameters are updated.

TABLE I
PARAMETERS IN EXPERIMENTS

Parameter	Value	Notes
$B(e), K(e)$	1Mbps, 2ms	
$L(p)$	512Byte	Specified for the UDP content size
Convolution Block	$(32, 8 \times 4)$	Transformer Convolution. Input 32, output 8, and 4 heads
Actor MLP	$(197, 2)$	For the "d3" variant. Input $5 + 32 \times 3 \times 2 = 197$, output 2
Critic MLP1	$(197, 8)$	Dimensionality reduction for each link
Critic MLP2	$(E \times 8, 64, 1)$	E is the number of links
c_1, c_2, c_3	$-1, 0.5, 0.001$	The coefficients in the PPO objective
lr	0.0001	Learning Rate
Topology	NSFNet	14 nodes and 20 bidirectional links
Train Steps	120,000	10 days, 24 hours, 5×100 steps per hour
Updates	12,000	10 updates per 100 steps

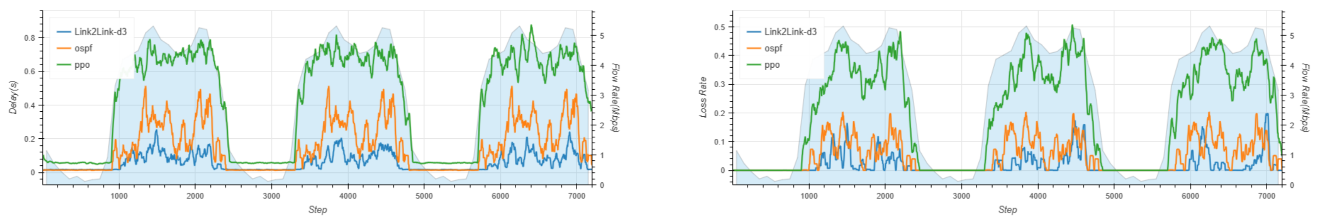

Fig. 5. Temporal Analysis of Delay and Loss Rate.

III. Performance Evaluation

In this section, we present and analyze the experiment results, aiming to answer the following questions,

- How does the performance of Link2Link compare to other algorithms and ablation variants?
- How robust is Link2Link across different network scenarios?

A. Experiment Setup

We constructed the simulation environment based on OMNet++ 5.6.3 [10] with the Inet library 4.2.1 [11], and implemented packet-level destination-differentiated probabilistic routing and a monitoring module to collect the traffic matrix as the state and QoS metrics as the reward. The agent is implemented using PyTorch, with ZeroMQ [12] facilitating interaction between the agent and the OMNet++ environment, enabling sequential reception of states and rewards.

The hyperparameters are listed in TABLE I. All link capacities are set to 1Mbps with a delay of 2ms. Each node randomly selects another node and generates UDP flows lasting for uniformly distributed periods. After one flow is finished, the node will select a new destination and generate a new flow. The traffic rate varies periodically over days, encompassing both low and high flow periods (refer to the light-blue area in Fig. 5).

The transformer convolution [13] is chosen as the GNN convolution layer. Each layer has a feature size of 32, an output size of 8, and 4 heads. Therefore, for a 3-layer convolution (”d3” variant), the input size for the linear layer is 197 and the output size is 2. The MLP parameters in the critic network are 197×8 and (E×8)×64×1, where E is the number of links, with E is 40 for the NSFNet topology (See Fig. 6).

B. Result and Analysis

To evaluate the performance of the Link2Link algorithm, we conducted training and testing under the NSFNet topology, comparing it with OSPF, a PPO algorithm using traffic matrix as state, and various ablation variants. A total of seven algorithms were compared, including:

- OSPF: the default implementation of the Inet library in the simulation environment OMNet++.
- PPO: a multi-layer perceptron with Proximal Policy Optimization, simply extended [4], with the input state being the traffic matrix for a specific destination defined as (23) and the one-hot encoding of the current destination.

$$S_{\text{matrix}} = \{u(d,e) \mid d \in D, e \in (N \times N)\}, \tag{23}$$

  where e represents the link between two nodes, and if no such link in the network, $u(d,e) = 0$.
- Link2Link d0,d1,d3,d5: variants of Link2Link with different number of GNN convolution layers. Among them, ”d0” is also an ablation variant that excludes GNN convolution layers entirely, directly forwarding the input state to the Actor’s linear layer.
- MLP: one ablation variant uses the same input state as Link2Link, that is, the link feature matrix, but utilizes a multi-layer perceptron instead of GNN convolution to process the link state.

Agents, excluding OSPF, are first trained in NSFNet topology, spanning a 10-day traffic cycle, totaling 120,000 steps (500×24×5) and 1,200 parameter updates. The evaluation lasts a 3-day traffic cycle with 7,200 steps (100×24×3). The results are presented in Fig. 5, where the light-blue area indicates network traffic levels and the flow rate refers

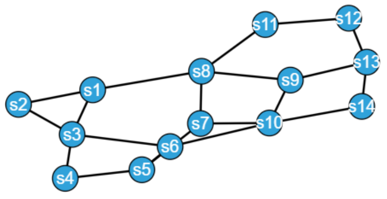

Fig. 6. The NSFNet Topology.

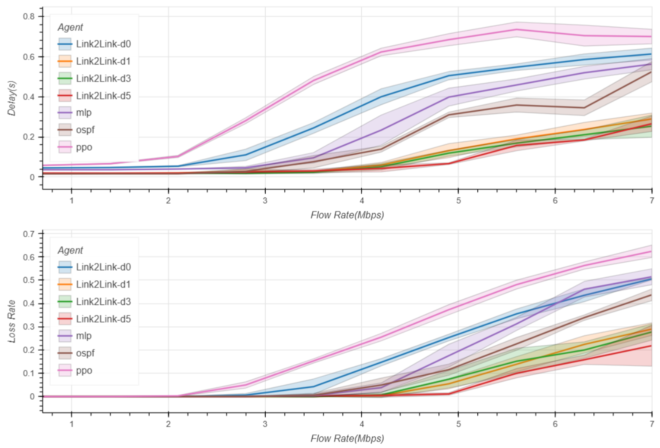

Fig. 7. Delay and Loss Rate Across Different Flow Rates.

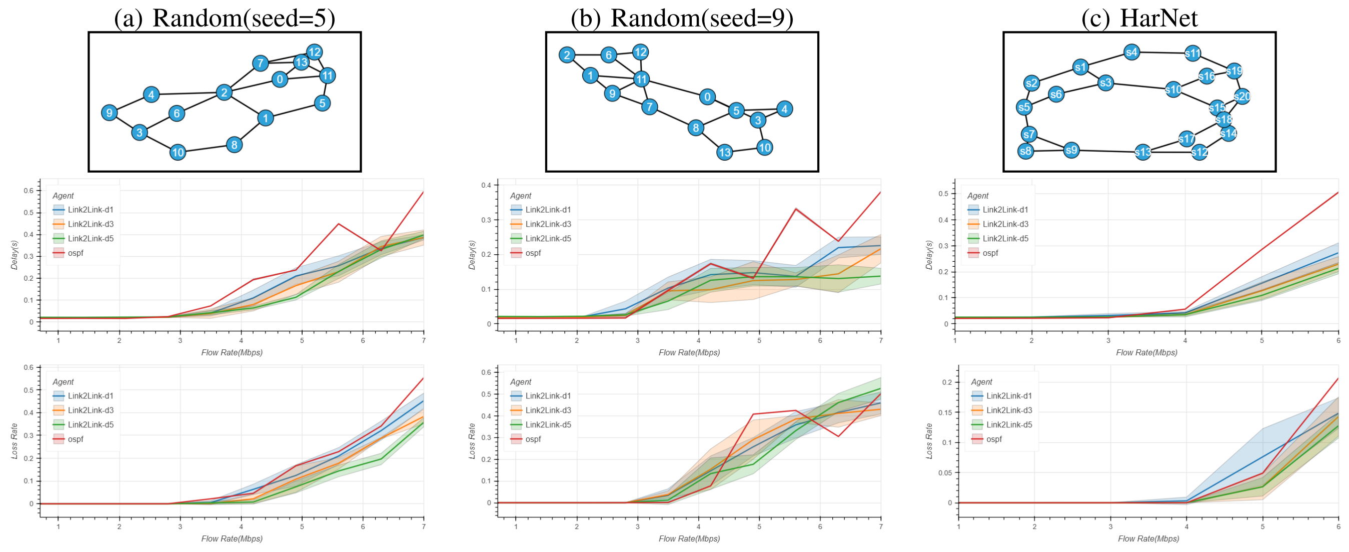

Fig. 8. Evaluating the Robustness of Performance Across Various Topologies

to the total flow demand. Only three algorithms are shown because some results are too close and will cause confusion. As illustrated, our proposed Link2Link generally matched OSPF in low traffic conditions and outperformed OSPF during peak traffic periods both in delay and loss rate. At low flow rates, both OSPF and Link2Link have delays under 20ms with low flow rates, while the PPO algorithm works with no loss but underperforms with a 30ms delay. At higher flow rates, Link2Link's average delay is only 100ms while OSPF's is 300ms and PPO's is around 700ms.

To better compare the performance of the algorithms in different flow rates, we tested them under varying flow rates, running 100 steps at each flow rate and repeating tests five times. The results are shown in Fig. 7, where the shaded area denotes the standard deviation from repeated experiments. When the flow rate is less than 2Mbps, all loss rates are 0, and the delay is around 20ms, while Link2Link maintains a 0.1s delay and 5% loss rate compared to OSPF's 0.3s delay and 10% loss at 5Mbps. This comparison more clearly shows that our algorithm surpasses OSPF, while OSPF outperforms PPO.

Moreover, comparing MLP with Link2Link (d1/d3/d5) proved that GNNs can process link features better than using multi-layer perceptron, while comparing MLP with PPO reveals that utilizing the link state design is more effective than directly employing the traffic matrix and destination one-hot encoding. Furthermore, the comparison of variants of different layers (d0/d1/d3/d5) indicates that using local link features alone (d0) is insufficient for solving routing problems, introducing only one GNN convolutional layer (d1) significantly enhances performance, and deeper GNN convolution layers (d3/d5) further improves performance, although the cost of slightly increased computational overhead. Interestingly, the variant with only one layer also performs well, which implies that each node merely requires the exchange of status information with its neighboring nodes, thereby fulfilling the prerequisites for the algorithm's distributed execution and consequently augmenting deployment adaptability.

To assess the robustness of the Link2Link algorithm, we utilized models trained under the NSFNet topology and tested them across different topologies to simulate changes in network topology, comparing the results with OSPF. Three topologies were employed in total: two were randomly generated 14-node, 20-link topologies akin to NSFNet, and the third was a larger topology, HarNet, with 20 nodes and 25 bidirectional links. For the random topologies, we used the ”gnm_random_graph(14, 20, seed=seed)” function from NetworkX [14], testing seeds from 0 to 9. After eliminating disconnected graphs and those with hanging nodes, only seeds 5 and 9 met the criteria, hence their selection for use. In this experiment, MLP and PPO agents were excluded due to their fully connected layers, which process the entire network's features, requiring training specific to each topology to optimize performance.

As presented in Fig. 8, the experimental results demonstrate that the Link2Link algorithm is highly robust to topology changes, and maintains superior performance over OSPF in all topologies, whether of similar scale or larger. And Fig. 8(a,b) shows that OSPF may perform badly in some flow rates with random topology. In addition, Fig. 8(c) reveals that the 1-layer (d1) GNN convolutional layer underperforms on larger topologies, whereas the 5-layer (d5) variant exhibits marginally better performance than the 3-layer (d3) variant, indicating that the number of layers need be increased when topology scale is large. Besides, this indicates that a model trained on a smaller topology can be seamlessly applied to a larger one without the need for retraining, a significant advantage considering the higher costs associated with simulating larger network topologies.

IV. Discussion

In this section, we delve into the scalability of the Link2Link algorithm. Benefiting from link-level processing, Link2Link has very strong scalability, including state representation, different routing polices, and the inherent ability to extend to multiple agents.

First, various network states can be readily modeled as link states. For instance, the distance between nodes is more straightforwardly represented as a link feature rather than a node state, and an additional feature dimension could represent the ratio to maximum link capacity if link capacities differ. Device attributes like remaining battery power can also be transformed into two link features, representing the remaining power of the source node and the destination node. Flow characteristics, such as QoS requirements that are independent of nodes and links, can be incorporated as additional features, processed by an MLP layer, and integrated into the concatenated link state.

Moreover, the output actions can be repurposed for various applications. For instance, link weights can serve as weight parameters for OSPF or ECMP. In flow path selection routing algorithms, the weights can function as parameters within a weighted link graph, facilitating the generation of viable paths through the random sampling of directed edge subsets, which are then designated as routing paths for the current flow.

Furthermore, the agent in this study is conceptualized as a single agent, which aggregates the entire network's link states in a central controller node and disseminates routing policies to each switch node. However, according to the model structure, each link is processed in parallel, which means that it can be interpreted as a distributed-execution multi-agent system with neighbor information exchange. In deployment, each agent can be deployed locally at a switch node, requiring only the associated link states to compute the weights of outgoing links, where link states can be obtained through additional link state exchanges. Besides, according to the robustness experiment, we can train the agent in networks with smaller topologies and deploy it to larger networks not need retaining.

V. Conclusion

In this paper, we extended the probabilistic routing model to handle general scenarios involving multi-source and multi-destination traffic. in addition, We presented Link2Link, a novel DRL-based routing algorithm that focuses on link-level decision-making to address the robustness and adaptability challenges in modern dynamic networks. Extensive experiments validate the algorithm's superior performance and robustness. Notably, Link2Link maintains strong performance even on larger, unseen topologies without the need for re-training. These results highlight the potential of Link2Link as a highly effective and adaptable solution for next-generation network routing challenges.

In Future work, we will focus on experimentally verifying the algorithm's scalability like flow-level routing, experimenting with more topologies and traffic patterns, and optimizing network routing with different QoS requirements and differentiated links.

References

[1] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[2] S. Kaviani, B. Ryu, E. Ahmed, K. Larson, A. Le, A. Yahja, and J. H. Kim, "DeepCQ+: Robust and Scalable Routing with Multi-Agent Deep Reinforcement Learning for Highly Dynamic Networks," in *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, Nov. 2021, pp. 31–36.

[3] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, "DRSIR: A deep reinforcement learning approach for routing in software-defined networking," *IEEE Transactions on Network and Service Management*, vol. 19, no. 4, pp. 4807–4820, Dec. 2022.

[4] Y. Wang, Y. Xiao, Y. Song, J. Zhou, and J. Liu, "Deep reinforcement learning based probabilistic cognitive routing: An empirical study with OMNeT++ and P4," in *2023 19th International Conference on Network and Service Management (CNSM)*, Oct. 2023, pp. 1–7.

[5] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[6] X. Mai, Q. Fu, and Y. Chen, "Packet routing with graph attention multi-agent reinforcement learning," in *2021 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2021, pp. 1–6.

[7] C. Liu, P. Wu, M. Xu, Y. Yang, and N. Geng, "Scalable deep reinforcement learning-based online routing for multi-type service requirements," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 8, pp. 2337–2351, Aug. 2023.

[8] X. Li, Y. Xiao, S. Liu, X. Lu, F. Liu, W. Zhou, and J. Liu, "Gappo - a graph attention reinforcement learning based robust routing algorithm," in *2023 IEEE 34th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2023, pp. 1–7.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[10] A. Varga, "Omnet++," in *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.

[11] L. Mészáros, A. Varga, and M. Kirsche, "Inet framework," *Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem*, pp. 55–106, 2019.

[12] P. Hintjens, *ZeroMQ: messaging for many applications.* " O'Reilly Media, Inc.", 2013.

[13] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," 2021.

[14] A. Hagberg, P. J. Swart, and D. A. Schult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), Tech. Rep., 2008.